Analysis, design and implementation of the game Fútbol en papel and application and comparison of some classical and bio-inspired artificial intelligence techniques to generate automatic controllers

Summary of Bachelor Thesis

Author: Marcos Martín Pozo Delgado
Director: Gustavo Recio Isasi
Tutor: Yago Sáez Achaerandio

1. Introduction

1.1 Objectives

The main objective of this Bachelor Thesis is to implement a version of the game Fútbol en papel in Java and to make a comparison between the classical artificial intelligent techniques for turn-based games and techniques inspired in Nature. Another objective is to analyze the complexity of the game.

1.2 The game Fútbol en papel

The game Fútbol en papel is a turned-based strategy game which is played usually in a grid sheet with two pens of different color.

There is a board with typically 10x16 squares (10x17 nodes) with two goals in the center of the top and the bottom of the board, with three nodes out of the board. The top goal is the player 2 goal and the bottom goal is the player 1 goal. In addition, there is a center horizontal line.

Each vertex of each square is a node, and initially the wall and the center line nodes are marked.

Furthermore, the board has edge, arcs or links, which are the lines that connect a node with another one. Initially the wall and the middle line edges are marked. Each node has 8 edges, whenever the node is not in an extreme of the board (in this case there would be a edge connecting a node which another one out of the board, and this is not possible): up, down, left, right and another per each diagonal.

An edge becomes marked when a player move upon it. A node is marked if any of its edges is marked.

In Figure 1 a representation of a typical board before starting the game is shown. The thickest edges of the board are the marked edges.

The player 1 starts the game in the center of the center line of the board.

A player who has the turn can move to any adjacent node whenever the edge which connects both nodes is not marked. If the target node is not marked the turn of the player ends, whereas if the node is marked the player can rebound (move more times) to another adjacent node whose connecting with current node edge is not marked or not rebound if this is not properly for him. If the player rebounds and the target node is marked he can rebound another time, the number of rebounds is not limited. The turn ends when a player moves to a not marked node, he moves to a marked node but he prefers not moving, the player cannot move (every adjacent node is connected with the current node by a marked edge, in which case the player is blocked and he loses) or the target node belongs to a goal (if it belongs to the opponent's goal the player wins and otherwise he loses).

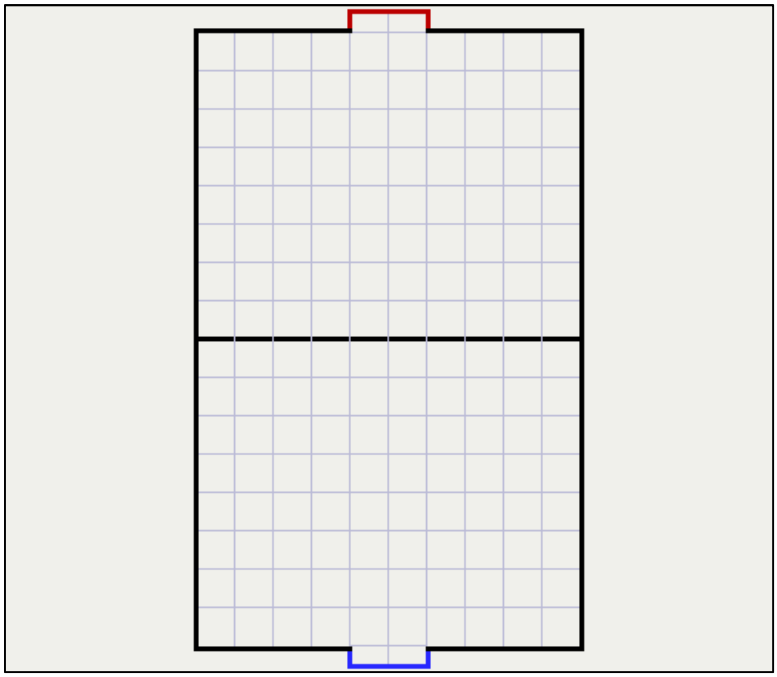

Figure 1: Typical board of 10x16 squares before starting the game

The **objective** of the game is getting a goal in the opponent's goal or making the opponent blocked (he cannot move to an adjacent node).

Next, an example of beginning of a game is described to illustrate better the game, since by text is complicated to understand it.

The game starts in the node (5, 8) with the turn for the player 1. In this situation the player cannot move left or right, as the edges of middle line are marked. Therefore, he could move up, down or any diagonal. Like the opponent's goal is in the top the logic movement is up, up-left diagonal or up-right diagonal. If we suppose that he moves up the resulting board is shown in Figure 2.

Since the target node (5, 7) is not marked the turn now belongs to player 2. The player 2 could move in any direction except down, as the edge below is marked. Nevertheless, moving up, up-left diagonal or up-right diagonal is not logic, since the opponents' goal is in the bottom. The best moves are down-left and down-right diagonals, in which the player could rebound. If we suppose that player 2 moves down-left diagonal, he could rebound up, down, up-left, down-left and down-right or not rebound, since the other edges are marked. If we suppose that player 2 moves down the resulting board is shown in Figure 3.

The game would continue until any player got a goal or any player got blocked.

In the implementation in order to permit not rebounding, each player has 4 seconds to rebound, and if in 4 seconds the player has not moved his turn ends.

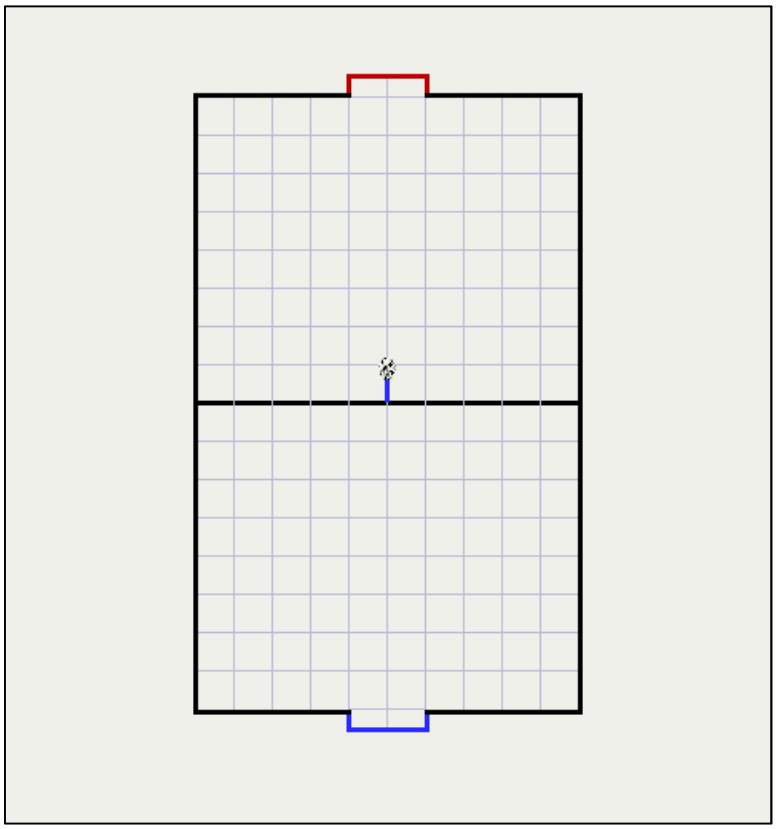

Figure 2: Board after player 1 moves up

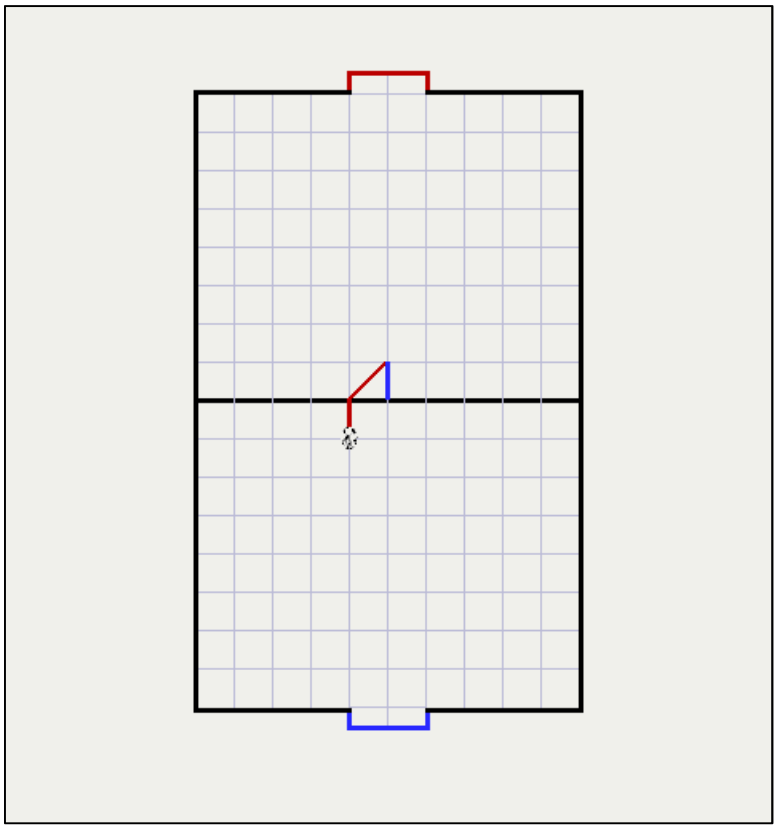

Figure 3: Board after player 2 moves down-left and down

2. Analysis of the problem

2.1 Analysis of the complexity of the domain

Due to the simplicity of the game, creating an automatic controller for the game Fútbol en papel could be thought easy. However, the fact that a player can move several times in the same turn due to the rebounds increases the complexity of the problem really much.

To study the complexity the algorithm used is minimax, a classic artificial intelligent algorithm in turned-based games. Minimax uses a search tree where each level is a turn of a player. This usually means that the depth of the tree is the number of turns evaluated.

Nonetheless, in this game a turn consists of several movements, so that in this problem the depth of the search tree is the number of turns multiplied to the maximum number of rebounds. In the game the rebounds are not limited, but in several runs of the game the maximum number found has been 35 rebounds, so that this number will be considered next.

In the worst case, the maximum number of movements from each node is 7 (8 except the edge that connect the current node with the anterior node). Thus, the depth of a search tree of two turns would be in the worst case is $35 \cdot 2 = 70$, and the number of explored nodes $7^{35 \cdot 2} = 7^{70} = 1.43 \cdot 10^{59}$ nodes.

The number of explored nodes in the worst case in a search tree of one turn (i.e., analyzing only the turn or the current player) is 7^{35}. If we suppose that the computer explores 10^{10} nodes per second (really it explores fewer nodes, roughly 10^7 or 10^8) the time required by minimax is 7^{35} nodes / 10^{10} nodes/second = $3.78 \cdot 10^{19}$ seconds = $2.88 \cdot 10^{25}$ years.

The time required is unreasonable. Due to this, the maximum number of rebounds explored was limited to 15 (this does not means that a player cannot do more movements than 15, but these possibilities are not explored in the search tree).

Furthermore, alpha-beta pruning, that reduces the exponent of explored nodes to a half in the best case, was used. In the average case, alpha-beta reduces the exponent of explored nodes to ¾.

In the average case, the possible movements from a node are 4 (1 in the best case, 7 in the worst case).

Thus, the explored nodes in the average case using alpha-beta pruning and limiting the rebounds to 15 are $4^{30 \cdot 3/4} = 4^{45/2} = 3.52 \cdot 10^{13}$ nodes. If we suppose again that the computer explores 10^{10} nodes per second, the time used to move is $4^{45/2}$ nodes / 10^{10} nodes/second = $3.52 \cdot 10^3$ seconds = 58.64 minutes.

This time, although lower, is remains unreasonable. However, since in fewer occasions doing 15 rebounds is possible the run time is lower than a minute. In the cases in what this time is bigger the search is limited to a minute with little loss of effectiveness.

2.2 State of the art

There is not any study about the game Fútbol en papel, but the game is included into the category known as turned-based strategy games. In this wide type of games in order to generate an automatic controller there are two basic options, depending on the complexity of the game.

In simple games, the most used technique is the alpha-beta algorithm or any of its improvements, like Negamax or NegaScout (for more information see [1]), combined with expert information or reinforcement learning or artificial neural nets. An example of alpha-beta and expert systems are Deep Blue [2], an example of reinforcement learning is the Samuel's checkers game [3, 4], and a sample of neural sets is Tesauro's Backgammon [5].

In the opposite, in complex games the most used technique is the scripts, also combined with expert systems or other optimization techniques like genetic algorithms. A sample of this is Webcard 3D y evolución de la inteligencia de alto nivel mediante algoritmos genéticos, a Final Project which can be got in [6]. Another technique used is the genetic programming, although is less used because it is necessary choosing good operators to be effective.

3. Design of the technique solution

The project is structured in Java packages. Next the content and functions or each package are analyzed.

3.1 The package futbolEnPapel2013

This package includes all other. In addition, it contains the interface and kernel classes.

The interface classes are `FutbolEnPapel2013`, `Instrucciones`, `Opciones` and `AcercaDe`. The class `FutbolEnPapel2013` is the main menu class. `Instrucciones` show the instructions of the game. `Opciones` contains the options (color of the board and level of each player). At last, `AcercaDe` contains the author, version and contact information.

The kernel classes are `Juego`, `JuegoObscuro` y `BuclePrincipal`. The class `Juego` contains the interface of the game and all information of the game. The class `JuegoObscuro` is a non-graphic version of `Juego`, essential to run a lot of games and evaluate the controllers. At last, `BuclePrincipal` is the main loop of the game, which is run in an independent thread.

3.2 The package fondo

This package contains the classes which permit having background images: `Fondo` y `FondoLienzo`. Both classes are `JPanel` that are included in objects `JFrame` and `JDialog`. The class `Fondo` is used to have a background image, whereas in the class `FondoLienzo` is also possible to paint lines (the lines of the players' movements).

3.3 The package estructura

This package contains the basic structures used by the game and the controllers: `Posicion`, `Nodo`, `Enlace` and `Tablero`. The class `Posicion` defines a position in the board. The class `Nodo` defines a node of the board and the class `Enlace` defines a edge. At last, the class `Tablero` defines a board, composed of nodes and edges.

3.4 The package jugador

This package contains the classes which move in each turn and the package controladores.

The class `Jugador` contains all information of a player and the methods used to move. The class `JugadorObscuro` is the equivalent to `Jugador` in non-graphic games.

The class `TrataTecla` is used to deal the key interruptions and the class `EscuchadorTemporizador` is used to deal the timer interruptions.

The class `Mover` does the movements of the controllers in an independent thread.

3.5 The package controladores

This package contains the classes used by the automatic controllers and the packages which contain these controllers.

The class `Grafo` contains a copy of the board and useful methods for the controllers.

The class `Controlador` is an abstract class which defines the properties of a controller.

3.6 The package alfaBeta

This package contains the controllers based in the minimax with alpha-beta pruning algorithm and the states used by this algorithms in the search tree.

3.7 The package hormigas

This package contains the controllers based in ant colony optimization and the package genetico.

3.8 The package genetico

This package contains the controllers optimized by genetic algorithms and the genetic algorithms classes.

4. Developed controllers

4.1 Alpha-beta

Firstly, requirements of minimax algorithm are described: the initial state of the alpha-beta is the current node in the board; the successors are the adjacent nodes which connecting edge is not marked; the evaluation ends when the maximum depth or goal is reached; the evaluation function is the distance to the opponent's goal, since also the A* algorithm was tried with identical results and much greater run time.

Secondly, the differences with the original algorithm are explained:

- **The own agent is the player MN instead of MAX,** since the evaluation function is the distance to the opponent's goal, the smallest is the best.
- **Maximum depth:** there is a maximum depth: the level of the controller (1-10).
- **A turn is composed for several movements:** to solve this problem the same method (`maxValor` or `minValor`) is called and the depth is reduced by 1 to not increment this (the depth is incremented in the beginning of the methods). The maximum number of rebounds is 15.
- **The connecting edge is marked during the recursion** to elude invalid movements.
- **If the child is the opponent's goal its value is 0 and if the child is the own goal its value is `INFINITO`.**
- **If the child is a blocked position its value is `INFINITO` if the method is `minValor` and `-INFINITO` otherwise.**
- **Only the longest path is considered,** to avoid expanding more nodes.

In addition, the states have an expansion order to do the alpha-beta pruning more effective. This order consists of the most probable good movements: the nearest states first.

In addition, due that the expansion order of a player is the opposite in another, the expansion order in `maxValor` is the contrary that in `minValor`.

1.1.1. Alpha-beta with costs

In order to improve the algorithm a version with costs was implemented. This version adds the edge's cost to the minimax value, but it is not necessary because this is done more effectively permitting rebounds when they are possible.

1.1.2. Alpha-beta without cycles

In order to improve the efficiency of the algorithm without limiting the maximum possible rebounds explored, a version that eludes cycles was implemented. Avoiding cycles can be good because when there are a lot of possible rebounds the most possibilities imply useless cycles.

Nevertheless, checking if the node already is in the path (to avoid cycles) consume many time, so that the improvement in time is little and the performance is lower due to avoiding cycles also avoid some good movements.

1.1.3. Dynamic alpha-beta

All anterior versions are static, all states exists in the beginning of the algorithm and only changes their minimax value. The problem of this version is that the minimax values of the states are overwritten by the execution of the algorithm due to the same nodes are evaluated in different times in the algorithm.

This version solves this problem creating the each node's children when it is explored by the algorithm. Thus, there are several states for the same position of the board, each corresponding to a different path.

This solution improves the performance really much, but it hast the inconvenient that it consumes a lot of RAM memory. With maximum depth 2 the memory consumed is roughly 600 MB.

Nonetheless, the nowadays computers have memory enough to run this algorithm, and its performance is much greater, so that this version is the chosen version to evaluate the other controllers.

4.2 Ant Colony Optimization

The ant colony optimization system implemented consists of the next points:

- It has 87 ants, number determined experimentally.
- During 15 iterations (number determined experimentally) the ants' path is cleaned and they move at the same time until they reach the opposite's goal. When every ant has reached the goal, they come to the nest, and the best ant comes two times. The nodes decision tables are updated every time every ant moves, whereas the pheromone is dispelled when every ant has come. When the 15 iterations end, the controller moves upon the edges which have more pheromone.
- The movements of the ants consist of getting the current node decision table and checking the taboo nodes. Next, the probabilities table is created from the decision table, setting to 0 the taboo nodes and normalizing the probabilities in order to they sum 1. Then, these probabilities are decreased by a heuristic value (0.87, got experimentally) multiplied by the distance to the goal and the probabilities are normalized again. Next, the accumulated probabilities table is created and the target edge is selected by accumulated roulette method. At last, the pheromone is added to the edge by 1 divided by the cost of the edge, and true value is returned if the ant has reached the goal and false value is returned otherwise.

- The return of the ants consists of adding 1 divided by the cost of the edge to every edge of the path pheromone.
- The dispelling of the pheromone is done multiplying the each edge pheromone by 1 - ρ (where the value of ρ is 0.4, since this is the usual value), and adding to this the new pheromone and setting the new pheromone to 0.
- The update of the nodes decision tables the next formula is used, where the α value is 1.15 to favor a bit the pheromone and the β value is 0.87 to disfavor a bit the cost:

$$\frac{(pheromone+pheromone_{new})^{\alpha}\cdot cost^{-\beta}}{\sum(pheromone^{\alpha}\cdot cost^{-\beta})}$$

This version has parameters which have been optimized by a genetic algorithm. The parameters are the number or ants, the number of iterations, the α parameter, the β parameter, the ρ parameter and the heuristic value that uses the distance to decrease the nodes probability.

1.1.4. FerDist version

Due to the previous version optimized has few ants (and the Ant Colony Optimization power is based in a lot of ants), this version was created. This version changes the pheromone update when the ants pass upon an edge. The pheromone added is 0.4 multiplied by the 1 divided by the edge cost plus 10 divided by distance to the opponent's goal. The value of 10 is due to the distances are bigger than the costs. In addition, has a tolerance value to not rebound in unfavorable situations (0.1).

This version also was optimized by a genetic algorithm. This genetic algorithm optimized the same parameters than in the previous version, the weight of the cost to deposit pheromone, the weight of the distance to deposit pheromone and the tolerance value.

1.1.5. FerDistRival version

This version was created to increase the performance of the previous version. This version adds a factor in the pheromone deposit. It consists of the cost of the edge multiplied by the distance to the own goal, in order to estimate the turns that the opponent needs to get a goal. To the previous formula to update the pheromone is added this value multiplied by 2.

This version also was optimized by a genetic algorithm. This genetic algorithm optimized the same parameters than in the previous version and the weight of the cost multiplied by the distance to the own goal.

1.1.6. Updating the nodes decision tables once per cycle

The two last versions are better than the first but they cannot win versus the alpha-beta controller. This is the reason that these versions were improved updating the nodes decision tables at the beginning of a cycle instead of when every ants move.

These versions also were optimized by a genetic algorithm and the last version was quite better than the previous versions.

4.3 Genetic algorithms

To optimize the ant colonies a genetic algorithm was used. This genetic algorithm has a tournament selection method, a simple cross method and a simple mutation method. In addition, the evaluation is parallel (is done in 16 in the initial generation and 8 threads in the rest) and consists of running 20 games and do the average. The population consists of 16 individuals and the intermediate population consists of 8 individuals. The replacement method consists of replacing the 8 worst individuals.

5. Evaluation

To evaluate the controllers many experiments were done. The results of these experiments are that the dynamic version of alpha-beta is the best version of alpha-beta far on the other versions. In addition, the FerDist version is better than the initial ant colonies version and that the best version of ant colonies is the FerDistRival version. At last, it is concluded that the alpha-beta controller is better than the ant colonies implemented.

6. Future work

Firstly, a genetic programming controller was begun, but it cannot be finished. In future work it will be ended.

Secondly, the Ant Colony Optimization controllers can be improved. In future work efforts will focus on improving the implemented controllers and developing other controllers through Ant Colony Optimization like a $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System or a *Rank-based* Ant System [7].

7. Conclusions

In this project an implementation of the game Fútbol en papel has been developed and also some artificial intelligence controllers have been implemented, and a comparison of these solutions has been done. Also an analysis of the domain has been done. Therefore, the objectives of the project have been reached.

At last, from the analysis of alpha-beta and Ant Colony Optimization controllers can be concluded that the alpha-beta controller is better than the Ant Colony Optimization controllers, and that these controllers can be improved.

8. References

[1] A. Reinefeld (1983). *An improvement to the scout tree-search algorithm. ICCA J. 6 (4) (1983) 4–14*.

[2] Campbell, Murray et al (2002). *Deep Blue. Artificial Intelligence 134 (2002) 57–83*.

[3] Samuel A. L. (1959). *Some studies in machine learning using the game of checkers*.

[4] Samuel A. L. (1967). *Some studies in machine learning using the game of checkers II. Recent Progress*.

[5] Tesauro, Gerald (2002). *Programming backgammon using self-teaching neural nets. Artificial Intelligence 134 (2002) 181-199*.

[6] Raúl Sampedro González (2008). *Webcard 3D y evolución de la inteligencia de alto nivel mediante algoritmos genéticos*.

[7] Marco Dorigo y Thomas Stüzle (2004). *Ant Colony Optimization*. The MIT Press.